Universidad Zaragoza
1542

Trabajo Fin de Grado

Plataforma de modelado y simulación de cuadricópteros con Modelica y PX4

Cuadricopter modeling and simulation platform using Modelica and PX4

Autor

Andrés Sánchez Pascual

Director

Enrique Teruel Doñate

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

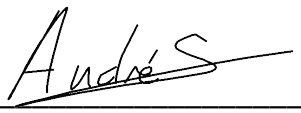
D./Dª. ANDRÉS SÁNCHEZ PASCUAL,

con nº de DNI 18173361A en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) PLATAFORMA DE MODELADO Y SIMULACIÓN DE MULTICÓPTEROS CON MODELICA Y PX4,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 28 de agosto de 2020

Fdo: 

AGRADECIMIENTOS

De corazón, Gracias.

A mi madre y a mi padre, por su apoyo, por su alegría, por sus sabios consejos. Por ser eterno ejemplo de constancia y sacrificio. Por hacerme brillar y haberme dado alas.

A mi hermano Pablo, por su incondicional apoyo, por ser fuente de inspiración, por ser mi mejor amigo, por todo lo vivido, por todas las risas. Eres luz.

Al resto de mi familia y amigos. Habéis compartido mis victorias y me habéis levantado en las derrotas. Me habéis ayudado, me habéis motivado, me habéis enseñado... pero sobre todo, me disteis vuestro tiempo, y ese, ese es el regalo más valioso de todos. Sois mi hogar.

A todos los profesores que durante mi vida se han esforzado por darme un futuro. A la Universidad de Zaragoza por su encomiable labor y, por supuesto, a mi director de proyecto Enrique Teruel que me ha ayudado a cerrar este ciclo.

RESUMEN

Los multicópteros se han convertido hoy en día en uno de los sistemas móviles autónomos con más tirón en el sector de la innovación e investigación. Debido a la complejidad de estos sistemas y los nefastos resultados que conlleva un error en su implementación, las plataformas de simulación han tomado un papel muy relevante como herramienta para el desarrollo de los mismos.

A lo largo de este proyecto se ha desarrollado una plataforma de simulación de multicópteros enfocada a su uso en centros universitarios donde tanto profesores como alumnos puedan desarrollar sus modelos de multicópteros y algoritmos de control de alto nivel y testearlos virtualmente con un sistema externo de control de vuelo.

En esta plataforma se ha integrado una arquitectura distribuida de simulación *Software In The Loop* para la cual se ha utilizado PX4 como sistema de control de vuelo y el modelo virtual del multicóptero, generado en OpenModelica, como planta.

Para el desarrollo del modelo virtual del multicóptero y la conexión con PX4 se han desarrollado dos nuevas librerías en OpenModelica. La primera de ellas contiene los modelos virtuales de los componentes del multicóptero y del entorno, utilizados para realizar el modelo completo de la planta del sistema. La segunda librería implementa los protocolos TCP y MAVLink para la sincronización y comunicación con PX4.

Índice

Capítulo 1

INTRODUCCIÓN

1.1. Introducción

La historia de los multicópteros es dilatada, ya en 1920 Etienne Omnichen sorprendió al mundo con su innovador diseño de vehículo aéreo multirotor el cual pretendía solucionar algunos de los principales defectos de los helicópteros de la época como eran su inestabilidad y la poca eficiencia del rotor de cola. En sus comienzos, los multicópteros originales distaban mucho de los actuales. Estos sistemas se concibieron como vehículos de transporte, eran propulsados mediante motores de combustión y el control era manual y mecánico. La complejidad del sistema de control del vehículo, unido a la complejidad de diseño mecánico, su poca eficiencia, limitada capacidad de carga y nulas aplicaciones bélicas hicieron que el desarrollo de este tipo de vehículo avanzara al ralentí.

Durante el final del siglo XX y comienzos del siglo XXI, los avances de la tecnología y la técnica, principalmente en las áreas de computación digital, telecomunicaciones y control automático, unido a la explosión de la robótica a nivel mundial, hicieron de este tipo de vehículo un sistema idóneo para investigación y desarrollo. Sin embargo su coste de adquisición todavía era relativamente alto.

A partir de la segunda década de este siglo, el sobrecoste, principalmente asociado al desarrollo electrónico y de control de estos sistemas, se redujo enormemente gracias a la entrada en el mercado de placas de desarrollo hardware y software de código abierto, como por ejemplo Arduino. Esto, unido al movimiento "maker"provocó la reducción de costes de adquisición y desarrollo de estos sistemas y hoy en día los multicópteros están presentes en casi la práctica totalidad de universidades y centros tecnológicos de todo el mundo por su uso ya extendido como herramienta de trabajo e investigación.

1.2. Motivación

De la mano de nuevas tecnologías como la inteligencia artificial (IA), la visión por computador, la computación distribuida, etc han surgido nuevas aplicaciones y áreas de trabajo. La robótica colaborativa, rama que motiva este proyecto, desarrolla algoritmos de alto nivel que permiten que múltiples robots, en nuestro caso multicópteros compartan

información, cooperen y optimicen sus tareas para realizarlas con mayor rendimiento.

Como herramienta para el avance de estas áreas, surge la necesidad de desarrollar entornos virtuales donde testear y validar dichos algoritmos de forma controlada y segura, evitando la problemática inherente a las pruebas realizadas en el mundo real.

1.3. Objetivos

Este proyecto tiene como objetivo principal el desarrollo de una plataforma de simulación de multicópteros sobre la cual basar los futuros trabajos de investigación en el área de sistemas robóticos colaborativos. Para ello se desarrollará un sistema capaz de realizar de forma rápida y sencilla modelos físicos de cuadricopteros y de conectarse a un simulador externo real para desarrollar simulaciones con arquitectura *Software In The Loop*.

Se buscará además una aplicabilidad directa de esta plataforma como herramienta de apoyo a la enseñanza en asignaturas universitarias (simulación, sistemas automáticos, teoría de control, robótica...) mediante la cual, tanto profesores como alumnos puedan modelizar y testear sus diseños.

De este modo se definen los siguientes requerimientos:

- La plataforma debe utilizar únicamente software no propietario con licencia abierta con el objetivo de dotarle del mayor acceso posible tanto para estudiantes como para investigadores y profesores.
- La plataforma debe permitir crear y modificar los diferentes módulos que componen el modelo físico del sistema (cuadricoptero y entorno) con un lenguaje de programación basado en diagramas de bloques.
- La plataforma debe disponer de un mínimo repertorio de bloques de simulación (motores, chasis, hélices...) que, en forma de librería, den soporte para la creación de modelos más complejos.
- El modelo de simulación física del sistema debe poder ser exportado a C/C++ como un modulo ejecutable independiente mediante generación automática de código.
- El modelo de simulación física debe ser capaz de conectarse con el sistema externo de control de vuelo y realizar simulaciones "Software in the Loop".
- La plataforma debe poseer una arquitectura distribuida básica, permitiendo la ejecución de cada uno de los procesos en diferentes máquinas y asegurar una robusta comunicación y sincronización entre ellos.

- La plataforma debe ser fácil de usar en cuanto a instalación, puesta en marcha, y obtención de resultados. Además, la lógica de bajo nivel de comunicación y sincronización entre procesos, compilación, autogeneración de código, etc debe ser transparente para el usuario final o, en caso de que el usuario deba actuar sobre tales puntos, estos deben estar claramente explicado.
- Siempre que sea posible técnicamente y se disponga de tiempo, tanto los módulos de simulación como el sistema en su totalidad debe ser verificado. En caso de que no se pueda mediante procesos experimentales con drones reales se puede realizar una primera verificación con simulaciones en softwares diferentes como por ejemplo Simulink.
- Por último, la plataforma debe estar bien documentada y los principales bloques de simulación deben añadir una breve reseña de su modelado, características, fundamentos teóricos, etc.

1.4. Contenidos

Los contenidos presentados en este documento están organizados de la siguiente forma:

- El Capítulo 2 introduce los conceptos básicos de los sistemas de simulación, analizando su arquitectura y los componentes que lo integran. Se estudian brevemente las soluciones ya existentes de simuladores de cuadricopteros y las características que diferencian a nuestra plataforma de ellos. Por último se presentarán los programas y herramientas software seleccionados para desarrollar nuestra plataforma.
- El Capítulo 3 se centra en el modelado multifísico de nuestra plataforma. En él se evalúan detalladamente los principales elementos que componen el modelo de planta del multicóptero. Para cada componente se presentan los fundamentos teóricos, simplificaciones, validaciones, etc útiles para el desarrollo de los modelos y la implementación de los mismos en OpenModelica.
- El Capítulo 4 se centra en la integración del sistema de control de vuelo en nuestra plataforma. Primero se estudian las arquitecturas de simulación existentes, haciendo hincapié en el concepto de simulación *Software In The Loop*. A continuación se revisan las características del sistema de control de vuelo externo utilizado y los sistemas de comunicación y simulación que deberá incorporar nuestra plataforma para poder comunicarse con él.
- El Capítulo 5 tiene como objetivo la verificación del sistema en su conjunto. Para ello se desarrollan todos los pasos a seguir para poder realizar una simulación SITL sobre

nuestra plataforma. Con objetivo de ilustrar el proceso, se modelizará un cuadricóptero real y se añadirá a la biblioteca de OpenModelica para que futuros usuarios lo puedan usar como ejemplo.

- En el Capítulo 6 presenta las conclusiones y lineas de continuación del proyecto.
- Finalmente se incluyen las tablas y anexos que acompañan a este documento.

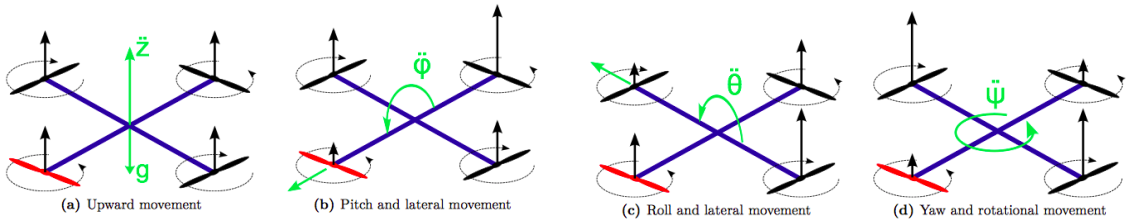
Capítulo 2

ARQUITECTURA DEL SIMULADOR

2.1. Descripción de un multicóptero

Un multicóptero es una construcción de vehículo aéreo compuesto por más de dos rotores. El control individual de la velocidad de giro de cada motor, de acuerdo a su disposición y sentido de giro, permiten la ejecución de un amplio rango de desplazamientos y giros.

Para el control fácil y eficiente de estos vehículos, los multicópteros de hoy en día poseen sistemas electrónicos digitales que se encargan, entre otras cosas, del manejo individual de cada motor. Estos sistemas facilitan la tarea del piloto e incluso dotan a la aeronave de capacidad de vuelo autónomo.



(a) Upward movement (b) Pitch and lateral movement (c) Roll and lateral movement (d) Yaw and rotational movement

Figura 2.1: Mecánica de movimiento del cuadricóptero

El vuelo autónomo de un multicóptero se realiza mediante un sistema de control de lazo cerrado en el cual están contenidos tanto la planta como los algoritmos de control y sistemas digitales anteriores. Esta arquitectura de control del multicóptero real deberá ser plasmada en su totalidad en nuestro sistema de simulación de vuelo, por lo que se considera útil presentar brevemente los principales elementos que la conforman.

2.2. Sistema de control lazo cerrado

Partiendo de su versión mas genérica, un sistema de control en lazo cerrado, representado en la Figura 2.2, esta compuesto por los elementos listados a continuación:

- Señal de entrada. Es la función objetivo que queremos que nuestro sistema alcance, su magnitud física, si la tiene, dependerá de como se haya modelado el sistema. En el caso de un UAV *(Unmanned Aerial Vehicle)*, puede referirse a ángulos de inclinación,

coordenadas geométricas, aceleración del sistema, etc.

– Señal de salida. Corresponde a los parámetros observados en nuestro sistema. Al igual que en el caso de la señal de entrada, pueden estar referidos a diferentes magnitudes físicas como inclinación del multicóptero, aceleración angular, altura, etc.

– Planta del sistema. Corresponde a la representación real del sistema bajo control. En el caso de un multicóptero, está compuesto por el chasis, los actuadores y sensores junto al entorno y sus efectos sobre el multicóptero.

– Controlador. Es el encargado de proveer de una acción adecuada al sistema para obtener la respuesta deseada. Para ello procesará la información proveniente de los sensores y la señal de entrada mediante el algoritmo de control y ejecutará las acciones necesarias en los actuadores para obtener el comportamiento deseado. En el caso de un multicóptero, el controlador es un microprocesador digital en el cual el algoritmo de control se integra completamente mediante software.

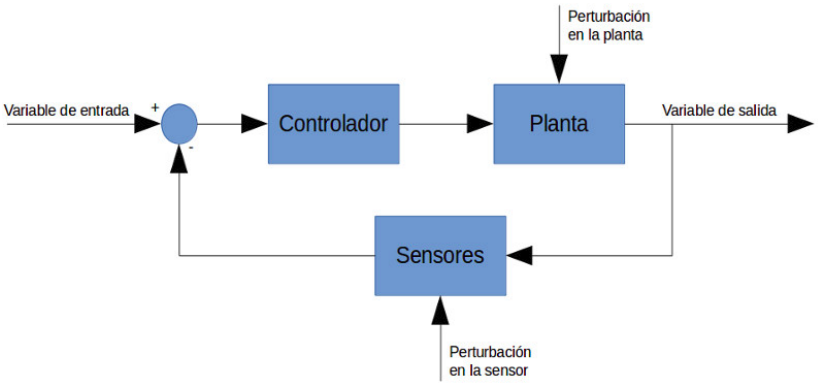


Figura 2.2: Sistema de control lazo cerrado genérico

2.2.1. Composición del simulador

En el simulador de un cuadricóptero, cada uno de los elementos anteriormente citados han de ser sustituido por sus correspondientes modelos virtuales. Así pues, se sustituirán todos aquellos elementos que conforman la planta, las perturbaciones externas y los sensores por sus correspondientes modelos virtuales desarrollados mediante software de modelado multifísico. Por otro lado, con el objetivo de desarrollar la simulación *Software In The Loop*, se eliminar áel hardware real del sistema electrónico de control de vuelo, simulando el comportamiento de los algoritmos de control sobre un ordenador estándar. (vease Figura 2.3)

2.3. Análisis de requerimientos

Un análisis inicial de los requerimientos permite vislumbrar las características buscadas en nuestro sistema. Este análisis sirve para la selección de aquellas herramientas que mejor se adapten a las necesidades del proyecto y, a la vez, nos otorga un criterio objetivo de validación del sistema una vez finalizado. La siguiente lista describe brevemente las características buscadas más importantes:

- **Software libre.** El sistema debe estar desarrollado a partir de herramientas de software libre de tal forma que cualquier módulo pueda ser estudiado, modificado, y utilizado libremente con cualquier fin y redistribuido con cambios y/o mejoras o sin ellas.
- **Multifísico:** El sistema ha de ser capaz de representar fenómenos físicos de distinta índole (eléctricos, mecánicos, fluidos...) todo ello simultáneamente en una plataforma unificada.
- **Distribuido:** El sistema debe permitir la distribución y ejecución de algunos de sus módulos en diferentes máquinas (ordenadores y/o microprocesadores) con motivo de aumentar la eficiencia de simulación, permitir ejecutar múltiples simulaciones a la vez, etc.
- **Modular y escalable:** La arquitectura del sistema debe realizarse de tal modo que sus módulos sean independientes y fácilmente intercambiables, aumentando la flexibilidad del sistema y su escalabilidad.
- **Exportable a código C/C++ estándar:** El modelo generado tiene que poder ejecutarse en maquinas virtuales, por lo que es necesario la autogeneración de código C o C++. En la mediada de lo posible se intentara utilizar el estándar POSIX de portabilidad de código.
- **Verificable:** Los requerimientos anteriormente citados deben poder ser fácilmente comprobados, estableciendo el cumplimiento o no de los mismos.

2.4. Estado del arte

El objetivo de esta sección es analizar las soluciones ya existentes para la simulación de multilcópteros que mayor sincronía presenten con nuestro desarrollo. Entre todas las estudiadas, destacan las siguientes:

- **Gazebo y MavSim:** Estas dos plataformas 'open source' permiten la simulación de cuadricópteros (entre otros) en un entorno virtual modelado mediante lenguajes de

programación formales (C++ y Java respectivamente). Ambos sistemas de simulación poseen composiciones de simulación integradas con la plataforma PX4 y su sistema de simulación "Software in the Loop". Entre sus virtudes están su velocidad de ejecución, su capacidad de extensión y bajo nivel de abstracción así como su portabilidad. Sin embargo, la modelización mediante este tipo de lenguajes de bajo nivel a menudo supone un problema en el desarrollo de sistemas de simulación complejos. La necesidad de conocimientos muy profundos del lenguaje así como la difícil visualización del sistema (frente a lenguajes visuales) hace de estos simuladores sistemas difíciles de entender, modificar y revisar.

- **Simscape:** Simscape™ permite crear modelos de sistemas físicos dentro del entorno de Simulink®, entorno de modelado visual de Matlab, referente en cuanto a software de simulación por su gran capacidad para el modelado de sistemas complejos. Entre sus ventajas se encuentran su interfaz gráfica en forma de bloques, su potencia de simulación, su extensa documentación y soporte por parte de Matlab y su comunidad, su extensa librería de componentes estándar, así como sus módulos de auto generación de código. La gran desventaja de esta plataforma reside en que se trata de software propietario (alejándose de los fines didácticos o de herramienta de soporte a la enseñanza buscados en este proyecto). Múltiples proyectos de desarrollo de simuladores existen para esta plataforma, aunque la mayoría son de uso privado.

Entre las soluciones existentes no hemos dado con ninguna que permita el modelado de cuadricópteros mediante el uso de lenguaje visual de bloques y software libre. Es por ello que creemos que el desarrollo de una plataforma que reúna estas características puede resultar muy interesante para la actividad de investigación y docencia.

2.5. Solución adoptada

El presente apartado describe las herramientas que mejor se adaptan a las especificaciones y objetivos del proyecto y que por tanto serán utilizadas para el desarrollo de nuestra plataforma.

2.5.1. OpenModelica

OpenModelica es un sistema de modelización y simulación de código abierto basado en el lenguaje de modelización estándar ***Modelica***, creado para su uso en entornos industriales y educativos. El lenguaje Modelica es un lenguaje no propietario, orientado a objetos y basado en ecuaciones, que está enfocado hacia el modelado de complejos sistemas de dominio

multifísico (mecánicos, eléctricos, electrónicos, hidráulicos, térmicos, control...).

Las principales herramientas que ofrece el frame de desarrollo OpenModelica y que lo hacen idóneo para el desarrollo del modelo multifísico del cuadricóptero son que permite la representación gráfica del sistema mediante modelos de bloques, que permite la generación y exportación de código C estándar a partir de código de simulación Modelica, que permite la generación de modelos según los estándares FMI y TLM para la simulación con módulos externos como FMUs, objetos TLM y modelos de Simulink, Adams, Beast y Modelica y, además, permite la ejecución dentro del modelo de funciones externas escritas en Python, C y C++.

2.5.2. PX4

PX4 es un software de control de vuelo de código abierto para drones y otros vehículos no tripulados. El proyecto ofrece un variado conjunto de herramientas para los desarrolladores de multicópteros para compartir tecnologías y crear aplicaciones a medida.

Entre las características más importantes para el desarrollo de nuestro proyecto cabe destacar que PX4 permite la simulación del controlador en sistemas operativos Linux, implementa las técnicas de simulación *Software In The Loop* y *Hardware In The Loop*, utiliza la librería de mensajería estándar MAVLink y es un sistema altamente documentado.

2.5.3. Comunicación entre procesos

Tanto el modelo del controlador como el modelo físico del cuadricóptero son procesos independientes que pueden estar alojados en la misma máquina o en máquinas diferentes y que se tienen sincronizar y comunicar.

Para dichas tareas se hará uso de los protocolos de comunicación sobre internet TCP/IP y el protocolo de mensajería estándar para vehículos autónomos MAVLink.

El ***protocolo TCP*** (Transmision Control Protocol) es es uno de los protocolos fundamentales en Internet. Esta fundamentado en el concepto de capas. Este protocolo, a diferencia de su pariente UDP, garantiza la entrega de los datos a su destino sin errores y en el mismo orden en que se transmitieron, lo que lo hace un protocolo fundamental en entornos de computación distribuida.

El ***protocolo MAVLink*** es un protocolo de mensajería ultraligero para la comunicación con drones y otros vehiculos no tripulados a través de mensajes estándar. Este sistema proporciona un sistema robusto para la serialización de los mensajes. Por otro lado, su toolchain ofrece una serie de herramientas para la autogeneración de las librerías MAVLink para cada uno de los lenguajes de programación soportados (C, C++, Python, Java, Swift...)

a partir de las definiciones XML de dichos mensajes.

2.6. Visión de conjunto de la plataforma

La composición final de nuestra plataforma de simulación, de acuerdo a las herramientas y programas citados anteriormente posee una estructura de lazo cerrado con los siguientes elementos:

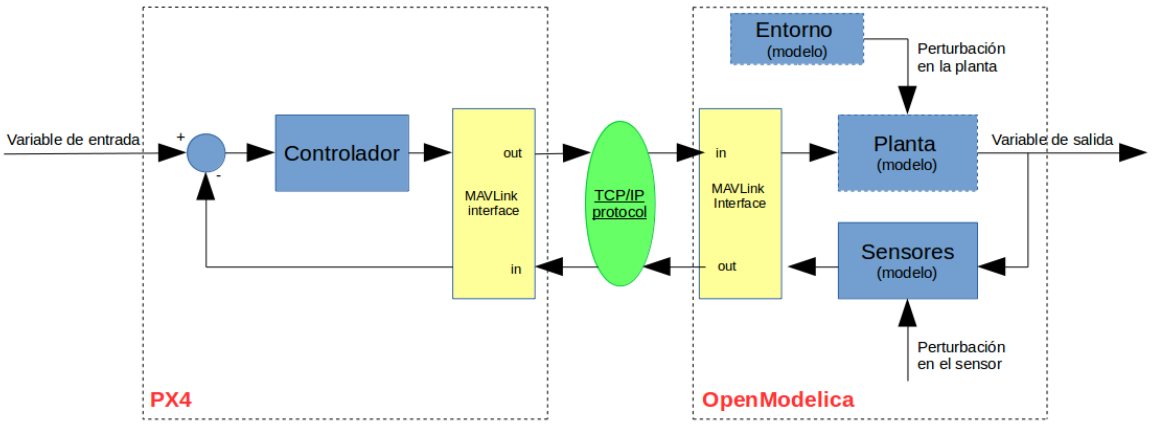


Figura 2.3: Descripción de conjunto de la plataforma

Capítulo 3

MODELO MULTIFÍSICO DE LA PLANTA

Como se vió en el Capítulo 2, todos aquellos sistemas físicos de nuestro cuadricóptero han de ser sustituidos por modelos virtuales realizados en OpenModelica, un software especializado de modelización y simulación de sistemas multifísicos.

A lo largo de este capítulo se presentan los modelos de los principales elementos constitutivos del multicóptero. De cada uno de estos modelos se especificaran los fundamentos teóricos, modelos experimentales, simplificaciones, etc, utilizados para el desarrollo de los mismos así como su implementación y uso en OpenModelica.

3.1. Modelización del multicóptero

3.1.1. Chasis

Mediante la librería Multibody[1] de Modelica se ha desarrollado un modelo parametrizado del chasis del cuadricóptero, considerando su estructura como un solido rígido.

Dado que la librería Multibody no admite el procesado de modelos 3D directo, el cálculo de las propiedades físicas de los sólidos como centro de gravedad, masa, momentos másicos de inercia, etc, ha de realizarse mediante software de diseño CAD (Solid Works, Inventor, Solid Edge...) como se describe en el Apéndice B. Estos parámetros deben incluirse mediante la ventana de parámetros de nuestro modelo creada para tal efecto.

La clase que modeliza el chasis del multicóptero y cuya representación gráfica se observa en la figura 3.1 cuenta con los siguientes componentes:

- **Masa central.** Masa total del chasis. Al tratarse de una masa puntual, carece de momentos másicos de inercia, por lo que estos se incluirán a través de la ventana de parámetros.
- **Brazos.** Los brazos del dron son elementos geométricos por lo que están exentos de propiedades físicas. Estos brazos, cuya masa y contribución a los momentos másicos de inercia ya están contemplados en el modelo mediante los parámetros CAD, permiten establecer el vinculo geométrico entre los motores y el chasis.

[1] `https://www.modelica.org/events/Conference2003/papers`

– **Masa del motor.** Masa individual de cada uno de los motores.

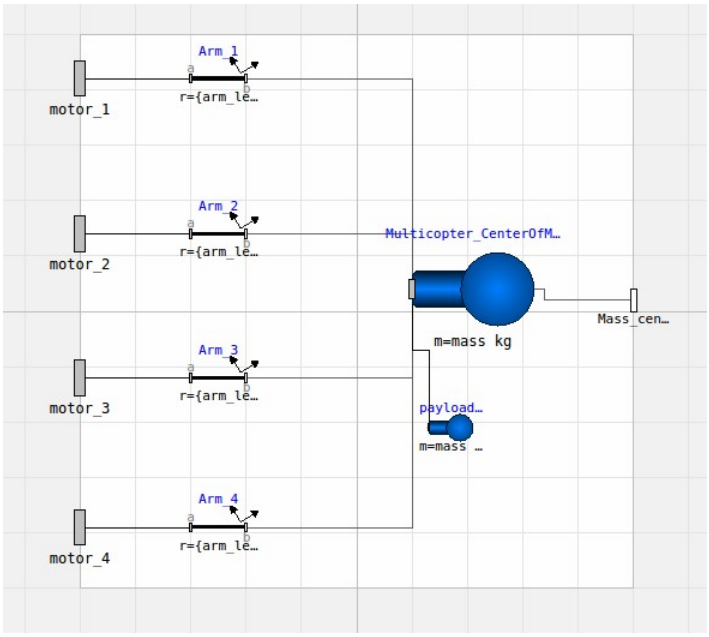


Figura 3.1: Modelo multibody del chasis

Este modelo permite la conexión con otros elementos de tipo mecánico mediante interfaces mecánicas de Modelica. Estas interfaces sirven para definir sistemas de coordenadas fijos a los elementos mecánicos a través de los cuales transmitir fuerza y par y definir movimientos. Así, uno de ellos es solidario al centro de masas y es utilizado para la imposición de sensores de aceleración y velocidad angular y el resto, modelizan la unión mecánica entre cada motor y el correspondiente brazo del multicóptero.

3.1.2. Sistema motriz

El correcto modelado del efecto de empuje y par generado por el conjunto de motor y hélices es difícil. Como se expone en [1], existen grandes dificultades para modelizar el comportamiento dinámico de los motores de tipo BLDC *(Brushless Direct Current)* usados en los drones. Para empezar, la relación entre la velocidad angular de la hélice y las fuerzas de sustentación y arrastre generadas son altamente no lineales y complicadas de describir matemáticamente. Por otro lado los ESC *(Electronic Speed Controllers)* que controlan estos motores son sistemas basados en microprocesadores digitales que, con sus propios parámetros de configuración, ejercen una gran influencia en el comportamiento final del motor. Además pocos o ningún fabricante especifican los parámetros internos de los motores y hélices por lo que es difícil realizar modelos matemáticos de estos.

Mientras que la caracterización estática de estos sistemas, como se verá a continuación, no

tiene mayor complicación, la caracterización del comportamiento dinámico debe realizarse, por lo general, mediante procedimientos experimentales, cuya recreación queda alejada tanto por complejidad como capacidad técnica del ámbito de este proyecto. No obstante, existen numerosos estudios y ensayos realizados en este ámbito y es posible utilizar dichos resultados en nuestro modelo simplificado.

Caracterización del comportamiento estático

La caracterización estática de estos sistemas es simple. Por un lado, podemos asumir, como se expresa en [1], que la relación entre la velocidad angular de la hélice y el empuje y par producido es estática. Numerosos fabricantes de motores suministran tablas que relacionan la velocidad angular con la potencia eléctrica y el empuje para diferentes combinación de motor y hélice. (Figura 3.2).

The voltage (V)	Paddle size	current (A)	thrust (G)	power (W)	efficiency (G/W)	speed (RPM)	Working temperature (° C)
11	EMAX8045	1	110	11	10.0	3650	
		2	200	22	9.1	4740	
		3	270	33	8.2	5540	
		4	330	44	7.5	6200	
		5	390	55	7.1	6700	
		6	440	66	6.7	7150	
		7.1	490	78.1	6.3	7400	36
	EMAX1045	1	130	11	11.8	2940	
		2	220	22	10.0	3860	
		3	290	33	8.8	4400	
		4	370	44	8.4	4940	
		5	430	55	7.8	5340	
		6	480	66	7.3	5720	
		7	540	77	7.0	5980	
		8	590	88	6.7	6170	
		9	640	99	6.5	6410	
		9.6	670	106	6.3	6530	43

Figura 3.2: Velocidad angular vs empuje y par

De este modo, la relación entre velocidad angular y empuje puede ser calcula directamente mediante interpolación cuadrática de acuerdo a los datos de Figura 3.2. Por su parte, el cálculo del par generado por el motor requiere algo más de esfuerzo. Es necesario en primera instancia convertir la potencia eléctrica a potencia mecánica de acuerdo al rendimiento del motor que se puede suponer según bibliografía en torno al 75 y 85 por ciento. En estado estacionario, la potencia mecánica generada por el motor es equivalente a la potencia disipada en forma de rozamiento por la hélice *(Drag Force)*.

En primera aproximación, podemos obtener el par generado por el motor de acuerdo a las expresiones siguientes:

$$P_{MECANICA} = \mu * P_{ELECTRICA} \tag{3.1}$$

$$P_{MECANICA} = T * \omega \tag{3.2}$$

$$T = \frac{\mu * P_{ELECTRICA}}{\omega} \tag{3.3}$$

Caracterización del comportamiento dinámico

En [1] se demuestra que, mediante procesos experimentales de identificación de planta (tanto en el dominio temporal como en el dominio de frecuencia), el comportamiento dinámico del conjunto motor BLDC, hélice y ESC puede modelarse con garantías mediante una función de transferencia de primer orden según la expresión:

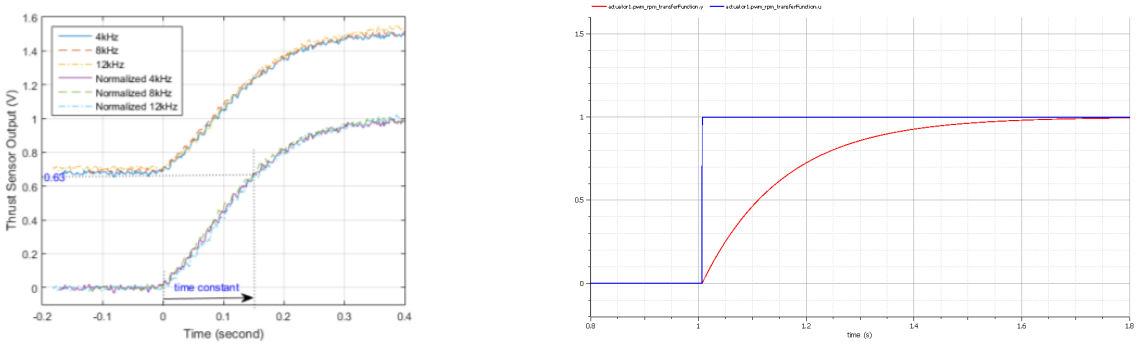
$$G(s) = \frac{1}{\tau s + 1} \tag{3.4}$$

En dichos ensayos, realizados sobre un motor y hélices similares a los de nuestro modelo, se obtiene el siguiente valor de la constante de tiempo normalizada:

$$\tau = 0{,}1499 \tag{3.5}$$

Dada la imposibilidad de replicar el ensayo de identificación de planta descrito en [1] para nuestro motor y hélice particular, se ha decidido implementar en nuestro modelo la constante de tiempo anterior, asumiendo la introducción de cierta imprecisión frente al sistema real.

La siguiente imagen representa la respuesta a escalón normalizada del motor real y virtual.



(a) Motor real

(b) Motor virtual

Figura 3.3: Comportamiento dinámico del sistema motriz

Modelo de OpenModelica

El modelo del sistema motriz en OpenModelica integra en un mismo bloque tanto el comportamiento estático como el comportamiento dinámico del conjunto formado por el motor, hélice y ESC.

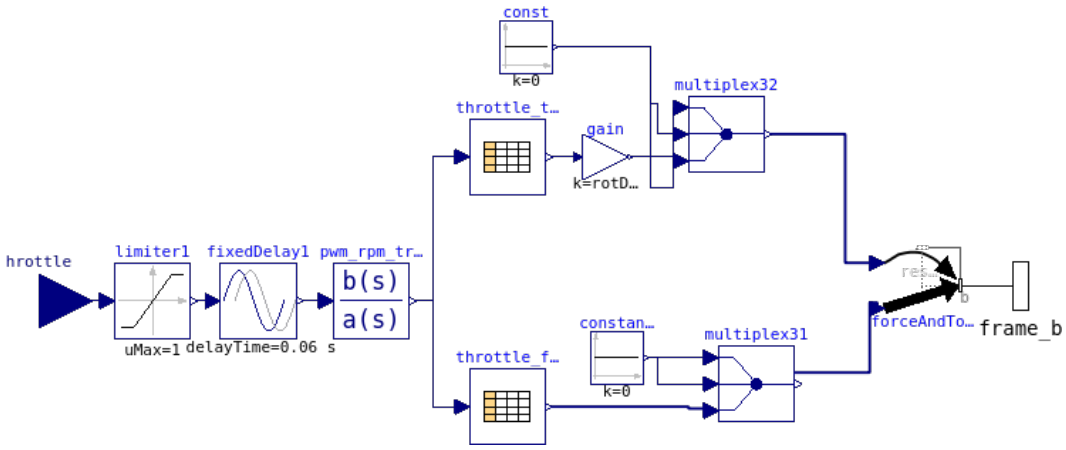


Figura 3.4: Modelo de sistema motriz

El comportamiento estático se incorpora en el modelo mediante *lookup-tables* de interpolación que incorporan los valores de empuje y par frente a velocidad angular. Por su parte el comportamiento dinámico se modeliza a través de un bloque estándar de función de transferencia donde se incluye la expresión matemática anterior (Ecuación 3.4) y un bloque de función de retardo que modela el tiempo de respuesta de los ESC (en torno a 16 Hz).

Este modelo posee una señal de entrada para establecer la potencia del motor en porcentaje (rango 0 a 100) y una interfaz de tipo mecánico para la transmisión de la fuerza y par generados al chasis.

3.1.3. Sensores

En un sistema de vuelo autónomo los sensores juegan un papel fundamental. Un sistema de control de lazo cerrado utiliza la información proveniente de los sensores para conocer el estado de la aeronave y actuar sobre esta.

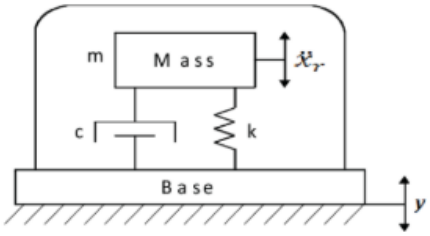
Los sensores son sistemas electromecánicos y por tanto los datos que recogen están influenciados por multitud de factores. El ruido electromagnético, el comportamiento dinámico del sensor, su resolución, el periodo de muestreo, por citar algunos de ellos han de ser correctamente representados en nuestro simulador pues en caso de omitir alguno de ellos, el resultado de la simulación diferirá enormemente de la realidad.

Sensor Inercial - IMU

Un sensor inercial o IMU (inertial measurement unit) es un dispositivo electrónico que mide aceleraciones lineales y velocidades angulares a las que esta sometido un cuerpo usando una combinación de acelerómetros y giróscopos.

Los drones llevan sensores inerciales de seis grados de libertad[2] correspondientes a las aceleraciones y velocidades angulares sobre cada uno sus ejes ortogonales. Mediante derivación e integración de estas medidas el controlador es capaz de calcular la posición, velocidad y aceleración de la aeronave.

Los sensores inerciales son sistemas piezoeléctricos que se deforman al actuar sobre ellos fuerzas externas (aceleraciones lineales o angulares), convirtiendo la deformación en variaciones de tensión eléctrica. Estos sistemas pueden modelizares como un sistema de masa, muelle y amortiguación mediante la ecuación de transferencia de segundo orden de la Figura 3.5.



$$G(s) = \frac{-s^2}{s^2 + (c/m)s + k/m}$$

Figura 3.5: Modelización de sensor piezoeléctrico inercial

A su vez, estos sistemas están compuestos de silicio, material caracterizado por una elevada rigidez y muy baja masa, por lo que la respuesta en frecuencia de estos sensores poseen una zona altamente lineal sobre la cual operan.

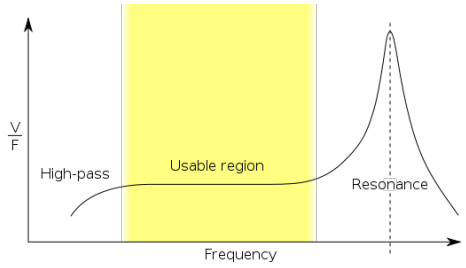


Figura 3.6: Respuesta en frecuencia de un sensor piezoeléctrico inercial

Este comportamiento permitiría eliminar el modelo de comportamiento dinámico del sensor que se considerara ideal dentro de la zona de uso del mismo (4 - 8000 Hz).

[2] A veces se consideran sensores inerciales de 9 grados de libertad cuando estos llevan incorporado el sensor magnético.

Modelo de OpenModelica

El modelo de sensor piezoeléctrico inercial desarrollado en OpenModelica representa cada uno de los ejes de los acelerómetros y giróscopos. Este modelo se ha ha realizado mediante tres bloques diferenciados, representando los diferentes componentes presentes en los sensor inercial SOC *(System On a Chip)* en los cuales además del piezoeléctrico existen otros sistemas digitales encargados de la configuración de filtros digitales, conversión ADC, codificación y envió de señales, etc.

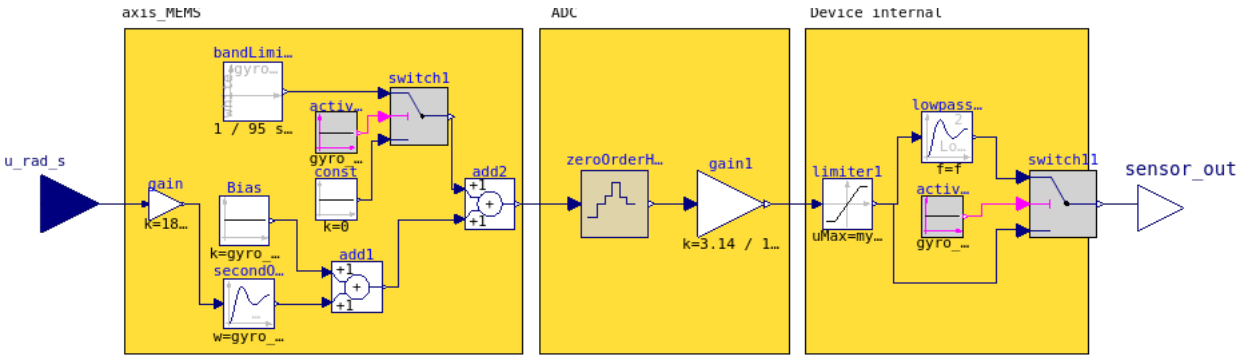


Figura 3.7: Modelo de piezoeléctrico en OpenModelica

A continuación se describe cada una de los diferentes bloques del modelo:

- Bloque 1. Representa el sistema electromecánico del piezoeléctrico donde se modeliza el comportamiento estático y dinámico. En esta etapa se modeliza el offset, el ruido electromagnético y la función de transferencia del comportamiento dinámico que, como vimos anteriormente, puede considerarse unitaria.

- Bloque 2. Caracteriza el efecto sobre la señal que tiene el sistema de adquisición y conversión ADC. En ella se puede ajustar la frecuencia de muestreo como se realizaría en un el sistema inercial real.

- Bloque 3. La última etapa representa el efecto del procesado digital del sistema. En esta etapa es posible configurar el rango de medición del sensor así como la frecuencia del filtro pasa bajo.

Los diferentes parámetros de este sistema (resolución, la frecuencia de muestreo, factor de escala, ganancia, ruido, etc.) son descritos en las hojas de especificaciones de los sensores inerciales comerciales y se añaden al modelo mediante la pestaña de configuración.

Por último, el modelo de sensor inercial de 6 ejes completo se obtiene mediante seis bloques individuales (tres de aceleración lineal y tres de velocidad angular), configurados de acuerdo a sus parámetros individuales:

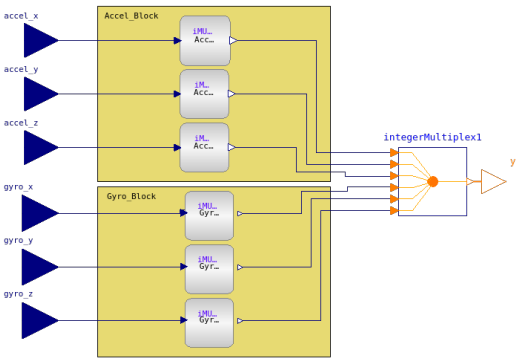


Figura 3.8: Composición de sensor inercial de 6 grados de libertad

Sensor barométrico

El sensor barométrico, debido a su sencillez y baja influencia en el sistema de vuelo se ha modelizado de forma sencilla mediante la adición de las correspondientes señales de ruido a la señal ideal de presión atmosférica.

3.2. Modelización del entorno

Con el objetivo de dotar al multicóptero de un entorno de vuelo se han representado y utilizado los siguientes modelos estándar:

- *1976 COESA-Extended U.S. Standard Atmosphere*. Modelo atmosférico con el cual calcular temperatura absoluta, presión y densidad del aire de acuerdo a la altitud geopotencial del multicóptero.

- *WGS84 Gravity Model*. Modelo gravitacional mediante el cual calcular la aceleración terrestre según la localización especifica usando el sistema geodético mundial (WGS84).

3.3. Modelo completo

Una vez desarrollados los modelos de cada componente del multicóptero por separado estos se ensamblan para obtener el modelo completo.

En un primer bloque se incluye el chasis, los motores, el bus de entrada de señales para el control de los motores (controlBus) y las señales de salida de los sensores ideales, obteniéndose el modelo de planta del multicóptero.

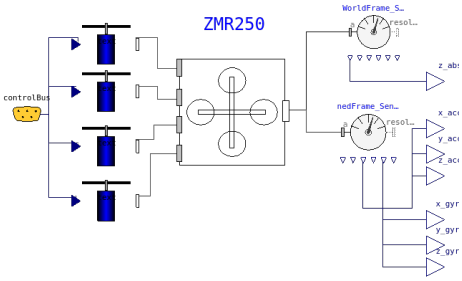


Figura 3.9: Modelo físico completo del multicóptero

Este modelo se incorpora en un nivel superior junto con los modelos de los sensores reales y los modelos de entorno, obteniéndose el modelo completo de simulación del multicóptero.

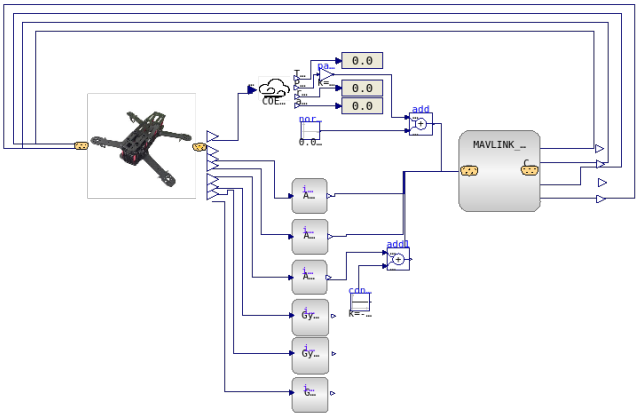


Figura 3.10: Modelo completo de planta y conexión con PX4

Un último bloque llamado MAVLink-TCP es utilizado para la comunicación con PX4. El desarrollo de este bloque queda reflejado en la Sección 4.3.

Capítulo 4

INTEGRACIÓN DEL CONTROLADOR

Como se vio en el capítulo de introducción, un sistema de control automático en lazo cerrado esta compuesto por la planta, el controlador, el entorno y las señales de entrada y salida según el siguiente esquema:

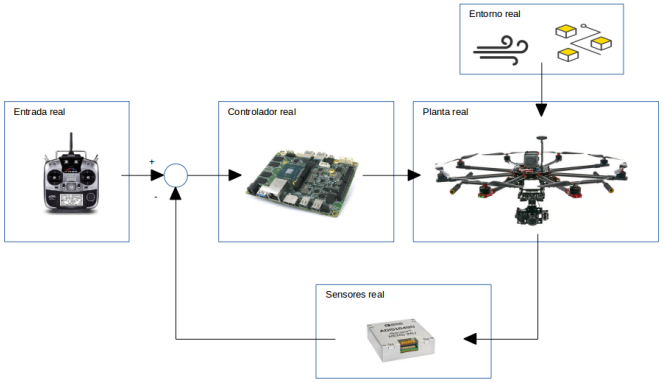


Figura 4.1: Sistema real

Según [2], el controlador se define como el elemento encargado de procesar la información disponible para poder generar una acción de control adecuada para el proceso. En el caso de un multicóptero el controlador es un microprocesador digital en el cual el algoritmo de control se integra completamente mediante software. A este microcontrolador están conectados los sensores y actuadores que le permiten percibir y actuar sobre el proceso, en este caso, el control de vuelo del dron.

Por su parte, los algoritmos de control utilizados para el control de estos vehículos distan mucho de ser un único sistema retroalimentado con parámetros y ganancias estáticas. Para cada tipo de vuelo soportado por el controlador (manual, estabilizado, loitter, GPS commanded...) un algoritmo de control diferente es integrado, ajustado e intercambiado en tiempo de vuelo. Así pues, recalcar que el objetivo, ni de esta sección ni de este proyecto, es recrear estos algoritmos sino que será el propio sistema de control, integrado externamente, el que aporte dicha funcionalidad.

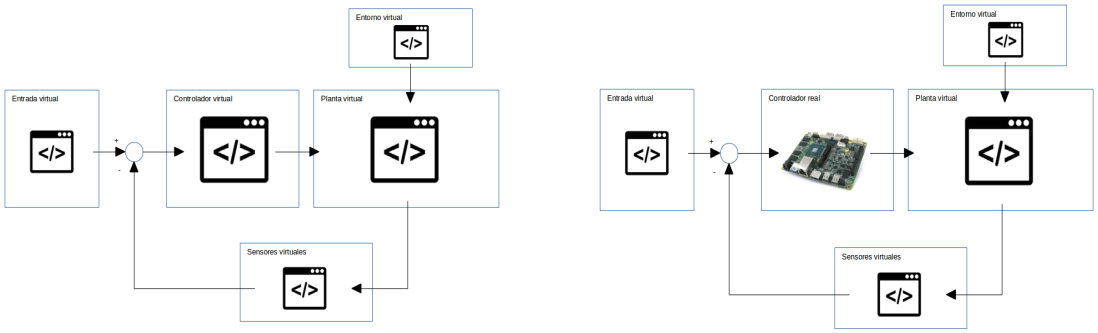
El objetivo de este capitulo es el estudio de los diferentes métodos de integración del controlador de vuelo, la justificación del controlador seleccionado en base a sus prestaciones

y la caracterización de los protocolos y sistemas de comunicación que debemos incorporar en nuestro sistema para llevar a cabo tal integración.

4.1. Integración del controlador. SITL vs HITL

Existen diversos métodos de realizar la integración del controlador y el simulador de la planta según sea su propósito y cada uno de ellos deriva en una arquitectura de simulador diferente. A continuación se detallan las dos más representativas:

- SITL (Software in the loop): Tanto el controlador (entendido como el algoritmo de control y el hardware) como la planta del proceso son simulados mediante software. En este caso el algoritmo de control es ejecutado en una maquina "virtual", con un hardware diferente al que se utilizará finalmente (Sistemas embebido). Este método permite acortar tiempos durante la etapa de desarrollo pues no hace falta tener definido por completo el hardware.
- HITL (Hardware in the loop): En este caso el algoritmo de control se ejecuta sobre el hardware del controlador real, mientras que el proceso controlado (planta) sigue siendo virtual. Así pues, las señales de los sensores y la acción de los actuadores que utiliza el sistema de control, provienen de la simulación del proceso. Este método es útil en etapas mas avanzadas de diseño, pues permite probar las capacidades reales del sistema de control (hardware y software real) sobre un entorno virtual, antes de pasar a las pruebas finales con el vehículo y el sistema de control reales.



(a) Software In The Loop (b) Hardware In The Loop

Figura 4.2: Arquitecturas de simulación

Dado que el propósito de uso principal de esta plataforma esta dirigida a investigación en áreas de robótica y apoyo docente no se ve práctico la utilización de una arquitectura HITL en esta primera versión. Por ello los esfuerzos se centrarán en la implementación de un simulador de tipo SITL.

4.2. Entorno PX4

PX4 destaca entre todos los proyectos de controladores de vuelo de código abierto disponibles en el mercado *(CleanCode, LibrePilot, Ardupilot...)* debido a sus enormes prestaciones, su robustez, su portabilidad y la enorme comunidad de desarrolladores que participan en su desarrollo.

PX4[1] es un software de control de vuelo de código abierto para drones y otros tipos de vehículos no tripulados. El entorno PX4 provee un conjunto de herramientas para desarrollar y mantener hardware y software para drones.

PX4 ha sido portado a diferentes placas hardware embebidas y permite que usuarios externos porten el código a nuevas plataformas por su cuenta. La característica más relevante que posee PX4 para este proyecto es su compatibilidad con sistemas operativos basados en UNIX. Esta característica permite que el código pueda ser compilado y ejecutado en ordenadores convencionales[2], en nuestro caso sobre el sistema operativo Linux.

PX4 es totalmente compatible con QGroundControl *(QGC)*, una aplicación que provee una base de control de vuelo y planificación de misiones de alto nivel de uso profesional y código abierto desde el cual manejaremos los cuadricópteros simulados en nuestro sistema.

4.2.1. Simulación en PX4

PX4 posee un módulo de simulación a través del cual implementar la arquitectura SITL. Con este modulo, hasta tres sistemas externos *(API/Offboard, QGroundControl y Simulator)* pueden conectarse e interactuar con el controlador, haciendo uso de los protocolos de red TCP/UDP y el protocolo de mensajería MAVLink.

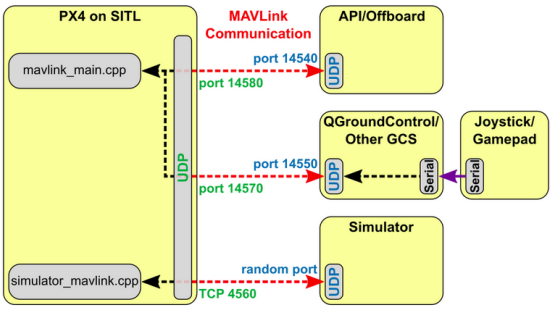


Figura 4.3: Modulo de simulación SITL de PX4. Extraído de `https://dev.px4.io/v1.9.0/en/simulation/`

[1]PX4 (`https://px4.io/`) es parte de Dronecode, una organización sin animo de lucro administrada por Linux para fomentar el uso de software de código abierto para vehículos voladores.

[2]PX4 es, por definición, un sistema de control en tiempo real. Cuando es ejecutado sobre sistemas operativos no *Real-Time* como Linux, esta característica se pierde.

Tanto el protocolo de red TCP como el protocolo MAVLink, descritos a continuación, son implementados en OpenModelica mediante la librería *"MAVLink-TCP"*, permitiendo la comunicación de nuestro modelo de multicóptero con el sistema PX4. Esta librería se crea desde cero ante la falta de soluciones preexistentes.

- **Protocolo de red TCP.**

  El protocolo de comunicaciones TCP/IP pertenece a la cuarta capa del modelo de comunicaciones OSI *(Open System Interconnection)*, encargada de efectuar el transporte de los datos (que se encuentran dentro del paquete) de la máquina origen a la de destino, independientemente del tipo de red física que esté utilizando.

  Entre las ventajas de este modelo cabe citar que es multiplataforma (permite multitud de sistemas operativos y hardware). Además, este modelo, basado en capas de información jerarquizadas, permite el seguimiento de los mensajes entre emisor y destinatario, asegurando que todos los mensajes enviados son recibidos por el sistema adecuado, en el mismo orden y sin errores.

  El protocolo de comunicaciones TCP sobre la capa de red de internet (IP) es la más utilizada hoy en día para comunicaciones entre ordenadores y es por ello que la práctica totalidad de sistemas operativos la incorporan como parte fundamental en el kernel.

  En OpenModelica, aprovechando que permite la llamada directa de funciones en Python, C y C++, la implementación de dicho protocolo se ha solventado mediante la programación, en C++, de un servidor TCP (TCP/IP Server) mediante el uso de sockets en Linux.

- **Protocolo de mensajería MAVLink.**

  MAVLink[3] es un protocolo de descripción y serialización de mensajes estándar para drones desarrollado por Lorenz Meier en 2009.

  Este protocolo define una gran variedad de mensajes estándar para la comunicación con sistemas autónomos y mediante su toolchain, permite la autogeneración de las librerías en diferentes lenguajes (C, C++, C, Python, Java...).

  La de API de comunicaciones de PX4 hace uso de un subconjunto de mensajes definidos en el protocolo de MAVLink para el intercambio de información con el sistema de simulación.

[3]MAVLink (`https://mavlink.io/`)

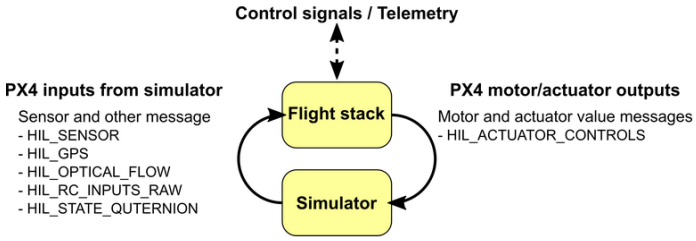


Figura 4.4: API de comunicacion de PX4. Extraído de `https://dev.px4.io/v1.9.0/en/simulation/`

En OpenModelica se hace uso de llamadas externas a las funciones C++ de MAVLink con lo cual la codificación y decodificación de los mensajes enviados es, para el usuario de OpenModelica, transparente.

4.3. Integración con OpenModelica

Como se vió en el apartado anterior, la comunicacion de OpenModelica con PX4 ha de realizarse mediante los protocolos de comunicaciones de red TCP-IP y el protocolo de comunicaciones MAVLink.

Dado que no existe actualmente ninguna librería que implemente esta funcionalidad en OpenModelica, parte de este proyecto se ha dedicado a la creación de la librería *MAVLink-TCP*, diseñada para permitir la conexión y sincronización con PX4 de forma intuitiva mediante el uso de bloques.

Podemos dividir la librería en dos partes a las que llamaremos *"Front-end" y "Back-end"* en referencia al lenguaje utilizado en desarrollo web.

- El front-end está implementado en OpenModelica y es la parte del software que interactua con el usuario, permitiéndole, mediante la adición de un bloque, dotar a su modelo de toda la funcionalidad de esta librería. El bloque *MAVLink-TCP*, con sus dos buses de comunicación (sensorBus y controlBus) permite el envío y recepción de los mensajes de los sensores y controles de los motores respectivamente con PX4.

- El back-end de la librería, invisible para el usuario, está implementado en C++ y actúa de puente entre OpenModelica y la implementación de los protocolos anteriormente citados. Esta parte de la librería es la encargada de establecer la conexión cliente-servidor con PX4, codificar y decodificar los mensajes MAVLink y gestionar la sincronización de las simulaciones mediante el sistema *Lookstep* descrito a continuación.

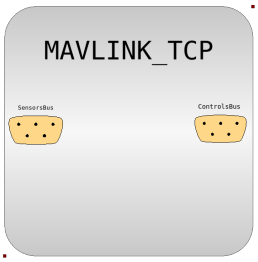


Figura 4.5: Bloque MAVLink_TCP

4.3.1. Simulación Lookstep

La sincronización de los módulos de simulación es importante pues de ello dependerá el comportamiento general del simulador y los resultados obtenidos.

El método *lockstep* desarrolla un sistema de simulación en el cual los módulos PX4 y el simulador se esperan el uno al otro en el envío de los mensajes de los actuadores y sensores, en lugar de funcionar a su propia velocidad. Con este sistema, además de sincronizar el tiempo de simulación de todos los modelos que participan, también se permite modificar la velocidad global de simulación (factor de velocidad de simulación) y permite parar o reanudar la simulación en tiempo de ejecución. Así, en la simulación *lookstep*, cada mensaje intercambiado entre módulos va acompañado de un registro temporal o *"timestamp"*. Este mensaje indica los valores obtenidos de la simulación junto con el tiempo de simulación en el que se han obtenido.

La secuencia de pasos es la siguiente:

1. El simulador envía un mensaje de sensor HIL_SENSOR que incluye una marca de tiempo *"timestamp"* para actualizar el estado del sensor y la hora de PX4.

2. PX4 recibe esto y hace una iteración de estimación de estado, controles, etc. y finalmente envía un mensaje de actuador HIL_ACTUATOR_CONTROLS.

3. La simulación espera hasta que recibe el mensaje del actuador / motor, luego simula la física y calcula el siguiente mensaje del sensor para enviarlo nuevamente a PX4.

4.3.2. Diagrama de interacción

El siguiente diagrama de interacción permite vislumbrar los procesos que intervienen así como el flujo de eventos y comunicación que se dan durante la simulación *lookstep* entre OpenModelica y PX4.

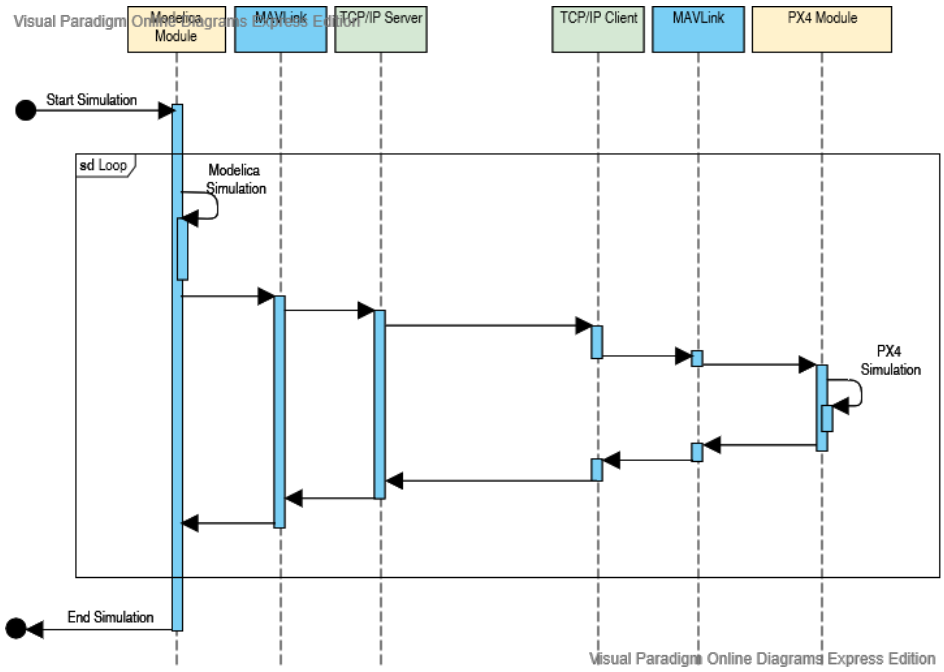


Figura 4.6: Diagrama de interacción OpenModelica-PX4

Capítulo 5

PRUEBAS Y RESULTADOS

El objetivo de este capítulo es comprobar que cada uno de los componentes de la plataforma han sido bien implementados y cumplen con los objetivos planteados. Para ello se ha decidido utilizar un procedimiento "demostrativo" en el cual se ha ejemplificado cada una de las fases que la puesta en marcha de la simulación conlleva.

En cada uno de los pasos se evidencian aquellos indicadores útiles para validar y/o verificar el correcto desempeño de diferentes partes del programa de tal modo que el usuario futuro sea capaz de realizar las mismas comprobaciones. Al final de este capítulo se muestra una tabla que relaciona los objetivos planteados y su consecución.

5.1. Configuración de simulación y exportación del modelo.

Tras finalizar el modelo completo de nuestro multicóptero en OpenModelica, el siguiente paso es configurar los parámetros de la simulación y exportar el modelo a lenguaje estándar C. De este modo obtendremos un módulo independiente que se podrá lanzar desde cualquier terminal en Linux

En el menú de configuración de la simulación impondremos un intervalo de simulación de 0.001s y seleccionaremos el tiempo de simulación según nuestras necesidades. Utilizaremos el método de integración *rungekutta* con una tolerancia de error 1e-3.

En opciones de OpenModelica seleccionaremos el formato de salida de nuestro ejecutable (C/C++), el gestor de compilación (GNU Make) y el compilador utilizado (clang clang++) y seleccionaremos la carpeta donde queramos que se genere nuestro modelo.

Tras estos pasos, si nuestro modelo carece de errores y compila correctamente estamos listos para lanzar la simulación.

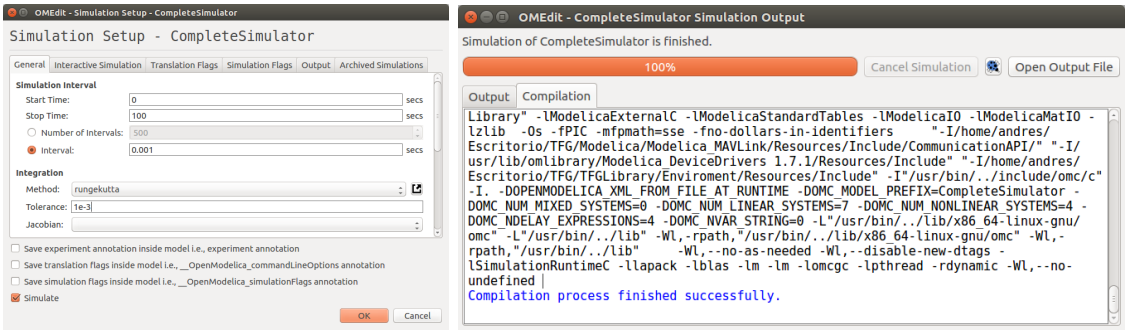


Figura 5.1: Configuración y exportación de modelo de planta.

5.2. Lanzamiento de módulos

Con el modelo de planta correctamente exportado y el modulo de simulación de PX4 preparado (vease Apéndice A), ambos pueden lanzarse como procesos independientes desde diferentes terminales de Linux.

El lanzamiento de PX4 ha de lanzarse primero. Para ello, desde la carpeta donde se haya clonado el repositorio se escribe el comando `make PX4_sitl none`. Esto lanzará un nodo cliente TCP/IP en espera de conexión con el servidor de nuestro modelo OpenModelica.

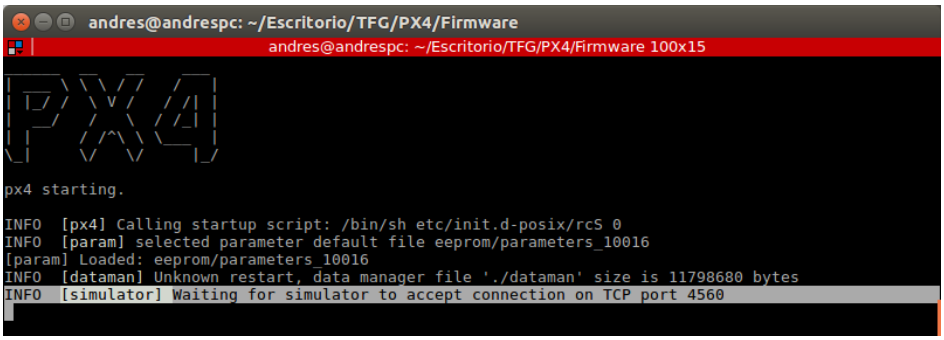


Figura 5.2: Módulo de simulación PX4 lanzado y a la espera de conexión TCP.

A continuación se procede con el lanzamiento del módulo de simulación de la planta. En una nueva terminal[1] vamos a la carpeta donde OpenModelica ha exportado nuestro modelo y escribimos el comando `./Nombre_del_modulo`.

La conexión entre ambos módulos es automática y se produce tras el lanzamiento del módulo de simulación de planta (TCP-Server).

[1]El lanzamiento de nuestro modelo de cuadricoptero OpenModelica puede realizarse directamente desde el ejecutable C exportado o directamente lanzando la simulación desde el entorno OpenModelica. Esta segunda opción tiene la ventaja de poder visualizar los resultados directamente una vez acabada la simulación.

5.3. Verificación de conexión TCP/IP

Si la conexión cliente-servidor mediante el protocolo TCP es exitosa, PX4 nos lo hará saber mediante el siguiente mensaje en el terminal *"Simulator connected on TCP port 4560"*. Una vez establecida la conexión, ambos procesos están sincronizados y comienzan a compartir mensajes de tipo MAVLink. El proceso de simulación ha comenzado.

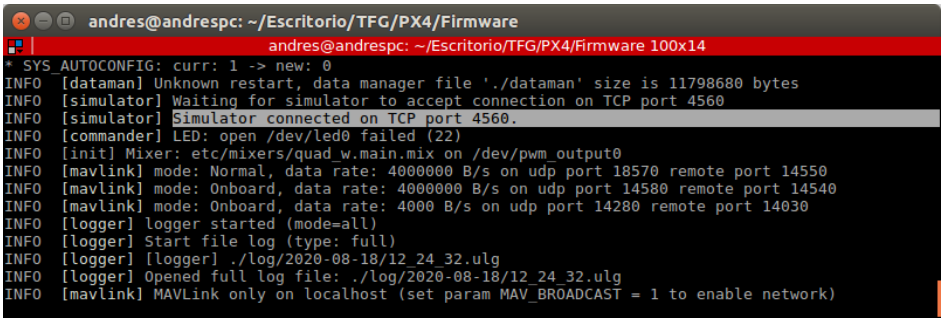


Figura 5.3: Conexión TCP establecida entre módulos.

5.4. Verificación de comunicación MAVLink.

El último paso para la verificación de la conexión hace referencia al envío y recepción de mensajes MAVLink. Si los mensajes intercambiados entre OpenModelica y PX4 están bien codificados y son procesados correctamente, veremos en la terminal de PX4, a modo de debug, la impresión por pantalla de algunos de ellos.

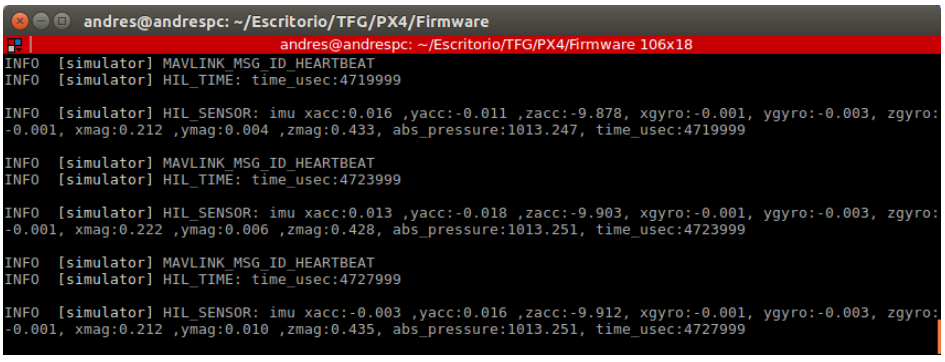


Figura 5.4: Envió de mensajes MAVLink entre módulos.

5.5. Envío de comandos de vuelo

Comprobamos también que somos capaces de enviar órdenes de vuelo de alto nivel a nuestro simulador mediante la estación base de control de vuelo QGroundStation.

En esta ocasión, pediremos a nuestro controlador de vuelo que despegue de manera autónoma con una velocidad de 2 m/s y, una vez alcanzada una altura de 10m, mantenga la posición.

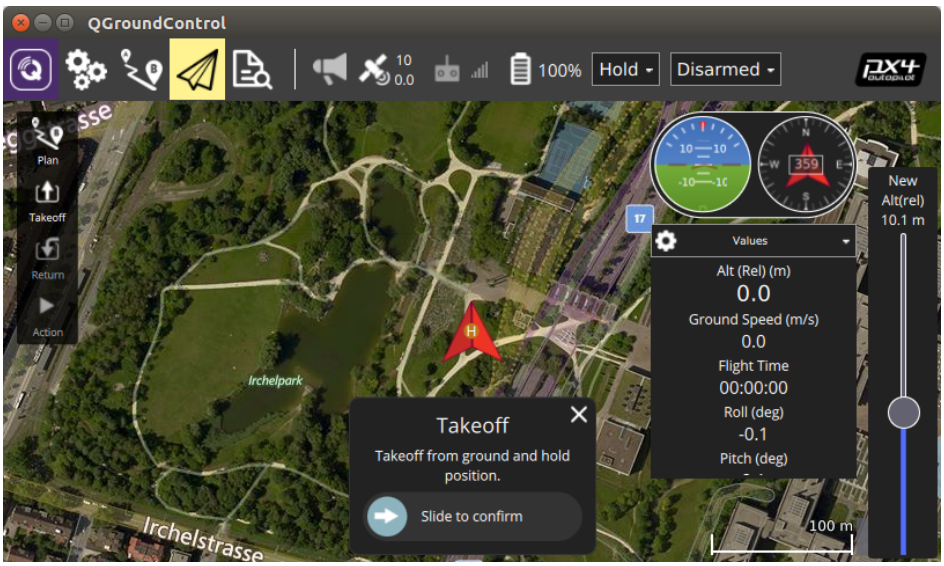


Figura 5.5: Envío de comandos de alto nivel mediante QGC.

El módulo de simulación multifísica genera un fichero con extensión Matlab (.mat) con el cual podremos revisar los resultados de la simulación de vuelo. Como ejemplo, en la imagen siguiente se grafica la altura alcanzada por nuestro multicóptero a lo largo de la misión.

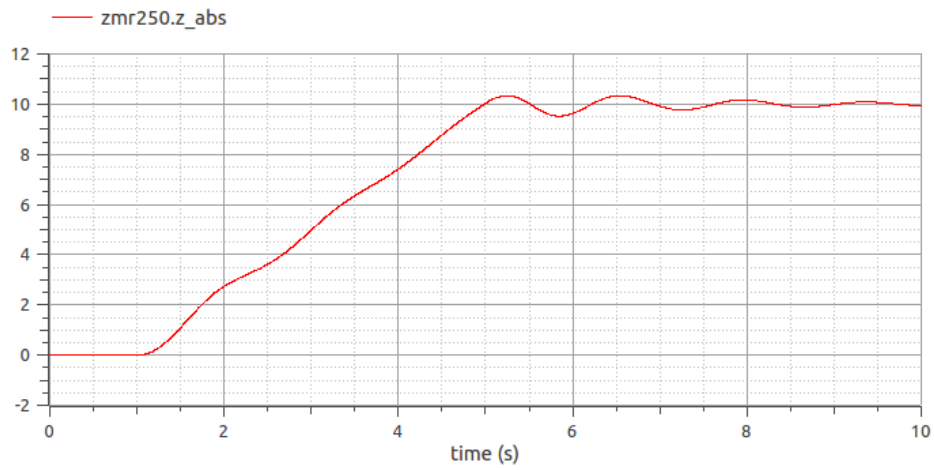


Figura 5.6: Perfil de altura de simulación SITL con PX4.

5.6. Visualización 3D

Una característica adicional de esta plataforma es la posibilidad que otorga OpenModelica de visualización 3D de los modelos. Aunque no es posible realizar una visualización de la simulación en tiempo de ejecución, sí que permite revisarla una vez esta ha finalizado.[2]

Así, la siguiente imagen muestra el aspecto de la visualización de la simulación anterior.

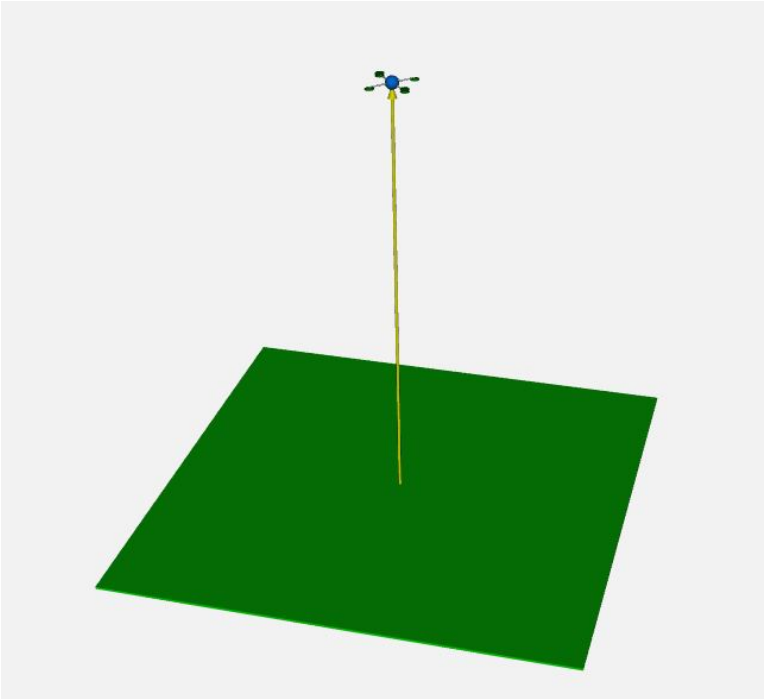


Figura 5.7: Visualización 3D de la simulación en OpenModelica.

5.7. Verificación y validación del sistema

La verificación es la disciplina del software cuyo objetivo es asegurar que el software satisface por completo todos los requisitos esperados. Los apartados anteriores, junto con el resto de capítulos, permiten valorar la consecución o no de los objetivos planteados al comienzo de este proyecto. La tabla Figura 5.8 resume dicha verificación.

[2]La simulación, como en el caso anterior genera un fichero *_visual.xml que se puede importar en OpenModelica para revisar la visulización.

Tipo	Requisito	Área aplicación	Descripción	Verificado
NF	Software Libre	Todo	Utilización de software no propietario con licencia abierta	Si
	Modelado multifísico	Modelo multifísico	El sistema ha de ser capaz de representar fenómenos de distinta índole (eléctricos, mecánicos, fluidos) en una plataforma unificada.	Si
	Lenguaje de modelado de bloques	Modelo multifísico	Utilización de lenguaje de modelado de bloques	Si
	C/C++ code generation	Modulo multifísico	Auto generación de código C/C++ del modelo de simulación	Si
	Modular y escalable	Todo	El sistema debe realizarse de tal modo que sus módulos sean independientes y fácilmente intercambiables, aumentando la flexibilidad del sistema y su escalabilidad.	Si
F	Arquitectura SITL	Todo	La plataforma debe disponer de una arquitectura SITL	Si
	Arquitectura distribuida	Todo	La plataforma debe disponer de una arquitectura distribuida, pudiendo ejecutarse cada modulo en maquinas diferentes.	Si
	Comunicación robusta	Todo	Los diferentes módulos deben establecer conexiones robustas para comunicación y sincronización entre procesos, haciendo uso de protocolos estándar para ello.	Si
	Comunicación transparente	Todo	La comunicación entre módulos debe ser transparente para el usuario el cual usara el modelo de bloques para su implementación	Si
	Controlador externo	Controlador	El sistema debe utilizar un controlador de vuelo externo	Si
	Visualización 3D	Modelo multifísico	El sistema debe ser capaz de representar el vuelo del modelo en un entorno 3D básico.	Si
	Revisión de resultados	Modelo multifísico	Deben autogenerarse ficheros con los resultados de la simulación de vuelo.	Si
	Liberia de componentes	Modelado multifísico	Se debe desarrollar una librería con los principales componentes conformantes de un multicóptero.	Si
	Documentación	Todo	La plataforma debe estar bien documentada, mediante guías, ejemplos de uso, etc.	Si

F – Funcionales
NF – No Funcionales

Figura 5.8: Verificación software de la plataforma.

Por otro lado, la validación del sistema es el proceso de revisión que corrobora que el sistema software producido cumple con las especificaciones y que logra su cometido. En el caso de un simulador, la validación consiste en determinar si el sistema modelado reproduce con la suficiente precisión al sistema real.

El proceso de validación de un modelo es complejo y laborioso que en particular requiere disponer de datos apropiados sobre el funcionamiento del sistema real. Puesto que el objetivo de este proyecto es la implementación de la plataforma de simulación y no el desarrollo de un modelo exacto de multicóptero, la parte de validación correspondiente se deja para futuros desarrollos con la plataforma. Sin embargo, de cara a efectuar esta validación se proponen, durante los apartados siguientes, las metodologías que se pueden utilizar para ello.

1. Validación frente a modelo experimental

Este método permite la utilización de un prototipo real de multicóptero con el que realizar la caracterización de los componentes y pertinentes ensayos de vuelo en laboratorio.

Los problemas asociados a este métodos son variados. Por un lado, la caracterización completa del vehículo real es compleja, cada elemento ha de ser caracterizado por separado en el laboratorio en un proceso por lo general, lento y tedioso. Estos elementos han de modelizarse con mucha exactitud en Modelica haciendo que el proceso de simulación sea muy exigente computacionalmente. Por otro lado, hay errores que en ningún caso podemos evitar con los medios disponibles para este proyecto. Por ejemplo, la implementación del controlador de vuelo en un sistema no Real-Time como Linux conlleva la pérdida de la perspectiva temporal, apareciendo retrasos en las tareas, inexactitud en los calculos, etc.

2. Validación frente a modelo virtual

Por su parte el método de comparación frente a otro modelo virtual supone la utilización de un segundo software de modelización multifísico con el que comparar los resultados obtenidos. Así, si se usan ambos controladores de vuelo idénticos, las diferencias de simulación recaerán únicamente en el modelo del cuadricóptero.

Como se vio en el Capítulo 2, Gazebo, Mavsim e incluso Simscape podrían utilizarse como modelos de comparación. El inconveniente de este método reside en obtener modelos paramétricamente comparables. En los simuladores descritos anteriormente, los modelos físicos de planta son modelados mediante ecuaciones escritas en C++ por lo que sería necesario una labor de acondicionamiento de los parámetros de estas ecuaciones para obtener modelos comparables.

Capítulo 6

CONCLUSIÓN Y LINEAS DE CONTINUACIÓN

A lo largo de este trabajo de fin de grado se ha trabajado en la implementación de una plataforma de simulación de multicópteros capaz de utilizar controladores de vuelo externos para efectuar simulaciones mediante arquitecturas Software In The Loop.

Con el objetivo en mente de su utilización como herramienta de apoyo a la docencia y en investigación en áreas de robótica, se le ha dotado de un carácter abierto, mediante el uso de herramientas OpenSource, como son OpenModelica, PX4 y MAVLink.

En OpenModelica se ha creado una librería personalizada de los principales componentes de un multicóptero, todos ellos fundamentados en conceptos teóricos y/o experimentales para su caracterización. A su vez, con el objetivo de probar la plataforma, se ha realizado un modelo completo y funcional de simulación, en el cual se han incluido un modelo del cuadricóptero comercial ZMR250, los modelos de sensores de una placa de vuelo comercial (Pixhawk 4) y el módulo de comunicaciones implementado ex profeso "MAVLink-TCP".

Por otro lado, se ha invertido gran parte del tiempo de este proyecto en indagar las caracteristicas y funcionamiento de la plataforma PX4, el protocolo MAVLink, los sistemas distribuidos y la comunicación cliente-servidor mediante TCP/IP. Con todo ello, se ha logrado implementar un sistema de simulación SITL con capacidad de ejecutar cada modulo en máquinas diferentes.

Con el objetivo de documentar el sistema, se han desarrollado una serie de Anexos en los cuales se explica el proceos de instalación de cada sistema, métodos de obtención de parámetros físicos para la caracterización del cuadricóptero, métodos de caracterización de sensores, etc, que esperemos sean de ayuda para el futuro desarrollador y/o usuario de esta plataforma.

Desde el punto de vista de alcance de objetivos marcados, se demuestra que se han alcanzado todos, a excepción de la validación del modelo de planta, el cual se deja como principal linea de trabajo para el futuro.

6.1. Trabajo futuro

Se han detectado varios aspectos que se deberán abordar para mejorar el funcionamiento de la solución actual:

- Validación del modelo. Queda pendiente la labor de validación del modelo general y de cada uno de sus componentes individualmente, ya sea vía comparación de modelos o vía experimental.
- Implementación de un sistema de visualización 3D real-time. Utilización de motores de visualización 3D como Gazebo, para visualizar el comportamiento del cuadricóptero y su entorno mientras se ejecuta la simulación y no una vez terminada, como ocurre en OpenModelica.
- Saturación de memoria. Es necesario realizar las modificaciones pertinentes dentro de OpenModelica para abordar la saturación de memoria RAM que se produce cuando se realizan simulaciones muy largas (mayor a 100 seg).
- Implementación HITL. Es interesante pensar en la implementación de la arquitectura de simulación *Hardware In The Loop* sobre placas de control de vuelo reales, con el objetivo de evitar los efectos que tiene utilizar en nuestra simulación sistemas operativos no Real-Time como Linux.
- Implementación de Interfaz Gráfica. El mantenimiento y lanzamiento de módulos es aún un poco confusa. Cada módulo se lanza por separado y es difícil de comprender para alguien sin profundos conocimientos de Linux. En una versión futura, cabría la opción de desarrollar una interfaz gráfica desde la cual lanzar los procesos de forma mas intuitiva.

Capítulo 7

BIBLIOGRAFÍA

[1] Myunggon Yoon. A transfer function model of thrust dynamics for multi-rotor helicopters. *International Journal of Engineering Research Technology (IJERT)*, pages 1 – 4, Jannuary 2016.

[2] Bolzern P., Scattolini R., and Schiavoni N. *Fundamentos de control automático.* McGraw-Hill/Interamericana de España, S.A.U., 2009.

[3] Samir Bouabdallah and Rolland Y. Siegwart. Full control of a quadrotor. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, December 2007.

[4] E.O. Doeblin. *Strumenti e metodi di misura.* Mc Graw Hill, 2005.

[5] Peter A. Fritzson. *Introduction to modeling and simulation of technical and physical systems with Modelica.* IEEE Press, 2011.

[6] Andrew Stuart Tanenbaum and Maarten Van Steen. *Distributed systems : principles and paradigms.* Pearson Educación, 2007.

[7] Daniel Pierre Bovet and Marco Cesati. *Understanding the Linux Kernel.* O'Reilly, 2005.

LISTA DE FIGURAS

ANEXOS

A. GUÍA DE USUARIO

A.1. Instalación y uso de PX4

El entorno de desarrollo de PX4 ofrece multitud de herramientas para desarrolladores de la plataforma. La guía de uso de PX4 `https://dev.px4.io/master/en/` reporta información detallada de la instalación y uso de cada una de ellas. A modo de resumen, los siguientes comandos permiten descargar, instalar y lanzar el modulo de simulación de PX4 en un ordenador Linux desde la terminal.

1. Instalación de la toolchain de PX4.

   ```
   ~$ mkdir PX4

   ~$ cd PX4/

   ~$ git clone https://github.com/PX4/Firmware.git --recursive
   ```

2. Compilacion y lanzamiento de la simulacion SITL en PX4

   ```
   ~$ cd PX4/Firmware/

   ~$ make px4_sitl none
   ```

A.2. Guía de uso de TFGLibrary

TFGLibrary es la librería desarrollada a lo largo de este proyecto para el modelado de los diferentes componentes presentes en el multicóptero y modelos básicos del entorno de vuelo.

Figura A.1: TFG Library

La librería, desarrollada con una arquitectura similar al resto de las librerías de Modelica, divide los componentes según ramas de modelado. Encontramos los siguientes:

- Airframes: Contiene los modelos electromecánicos del dron. En este bloque encontramos los modelos del chasis y los motores así como varios modelos reales preconfigurados coo el DJI_F450 o el ZMR250.

- Sensors: Contiene los modelos de los sensores. Entre ellos encontramos modelos de sensor de temperatura, sensor barométrico, sensor inercial y brújula.

- Buses: Contiene los buses de comunicación de datos entre los distintos componentes del multicóptero. Entre ellos están el bus de sensores (StatesBus) el bus de control de los motores (BusControl) y el bus de entorno (EnviromentBus).

- Enviroment: Contiene los modelos del entorno de vuelo. Entre ellos se incluyen el modelo gravitatorio WGS84 y el modelo atmosférico COESA_Athmosphere_Model.

- Util: Contiene elementos de utilidad para el desarrollo de los modelos.

A.3. Guía de uso de MAVLink-TCP

La librería MAVLink-TCP es la encargada de proveer un sistema de sincronización y comunicacion robusta con PX4. Esta librería esta esta dividida, siguiendo el lenguaje de desarrollo web, en la capa de interfaz de usuario en OpenModelica o "front-end" y la capa de acceso de datos o "back-end"

La arquitectura del "front-end" al igual que en el caso anterior responde a la arquitectura típica de librerías de Modelica. En esta librería encontramos los siguientes bloques:

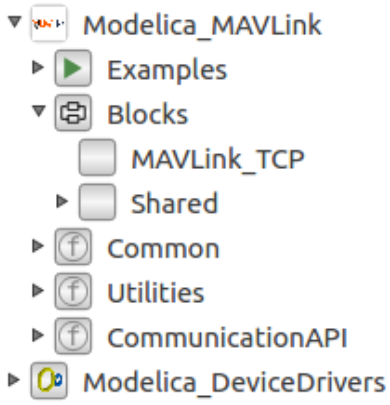


Figura A.2: MAVLink/TCP Library

El “back-end” por otro lado está implementado mediante ficheros C++ accesibles en la ruta `/Modelica_MAVLink/Resources/Include/CommunicationAPI/`. Entre ellos, destacan los siguientes dos ficheros:

- TCP.h. Fichero encargado de la sincronización entre procesos mediante el protocolo TCP-IP.
- ModelicaMavlinkWrappers.h. Fichero encargado de la codificación/decodificación de mensajes con el protocolo MAVLink.

B. CÁLCULO DE PROPIEDADES FÍSICAS

Como se vio en el capitulo Capítulo 3, la librería Multibody de OpenModelica permite el modelado y visualización de sistemas mecánicos tridimensionales complejos. Para ello, la biblioteca provee un gran numero de elementos mecánicos estándar, como uniones, frames de coordenadas, fuerzas, solidos estándar, etc.

Uno de los mayores defectos de esta librería es que no permite la utilización directa de solidos CAD en el modelo. Los solidos deben modelarse mediante elementos estándar (masa puntual, cilindro, esfera, barra rectangular, etc) y por tanto las propiedades físicas (masa, ejes principales de inercia, momentos de inercia, centro de masas, etc) han de añadirse al modelo manualmente a través de la correspondiente pestaña de propiedades.

El presente capítulo muestra al usuario el procedimiento para extraer estas propiedades de los sistemas de modelado 3D y la inclusión en nuestro modelo de chasis de OpenModelica.

B.1. Obtención de parámetros en CAD

La obtención de las propiedades anteriormente citadas es un paso sencillo en la mayoría de programas de modelado 3D del mercado. (Inventor, Catia, SolidWorks, etc).

En Inventor por ejemplo, una vez desarrollado el modelo de chasis de nuestro multicóptero podemos acceder a estos datos mediante **Archivo** → **iProperties** → **Propiedades Fisicas**

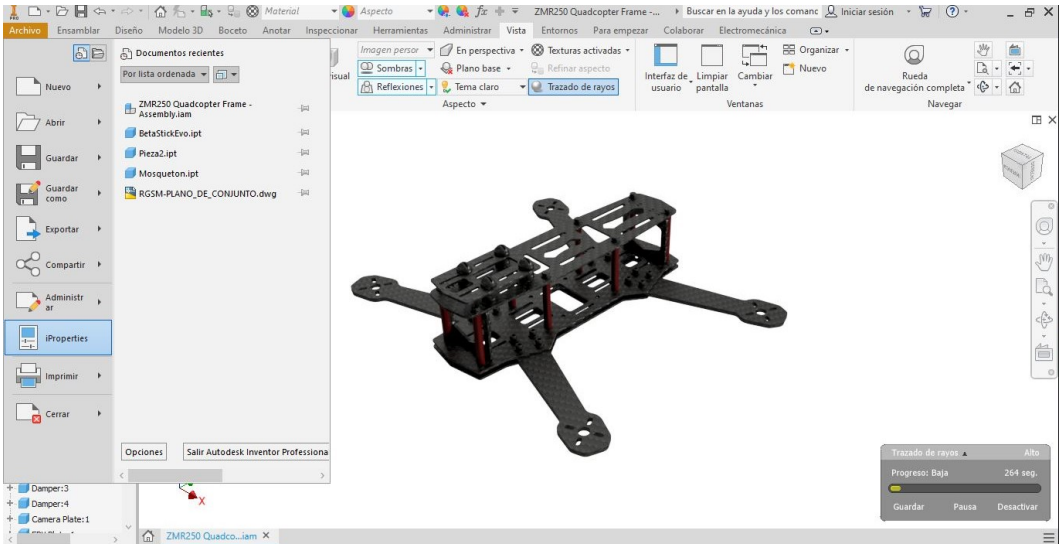


Figura B.1: Modelo tridimensional del ZMR250 en Inventor

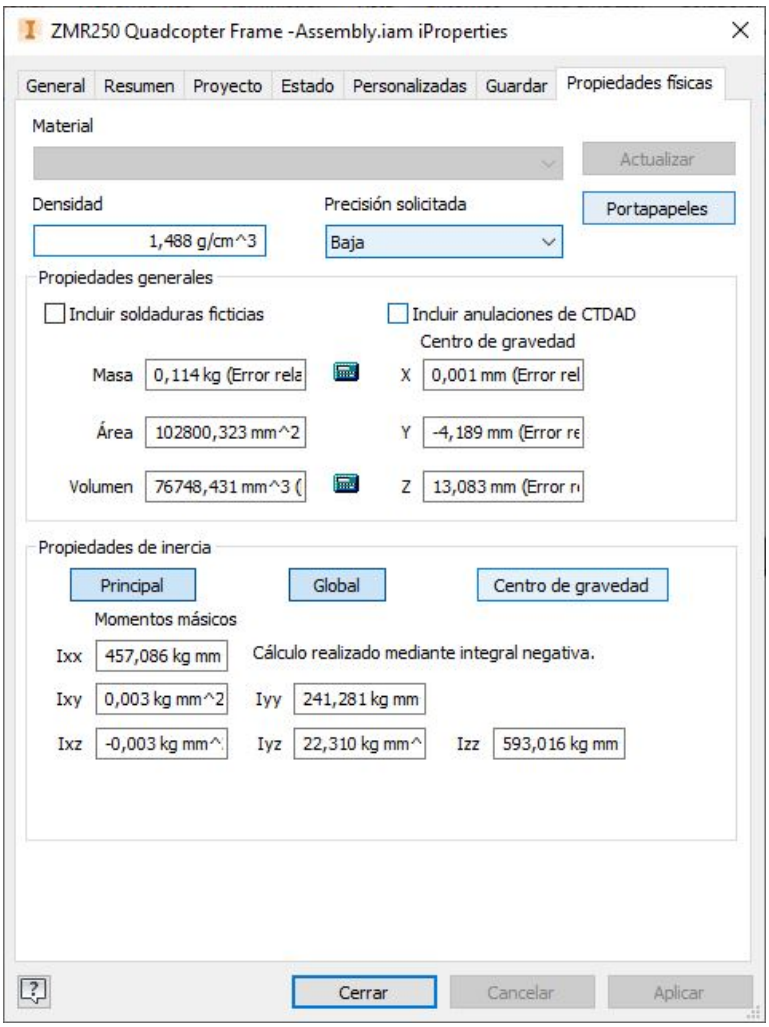


Figura B.2: Calculo de propiedades en Inventor

B.2. Inclusión de propiedades en OpenModelica

El bloque “chasis” de la librería de multicópteros de OpenModelica permite la introducción manual de las propiedades físicas del chasis mediante su pestaña de propiedades.

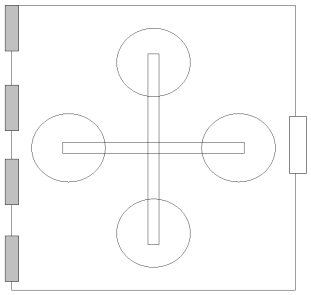


Figura B.3: Modelo de chasis en OpenModelica

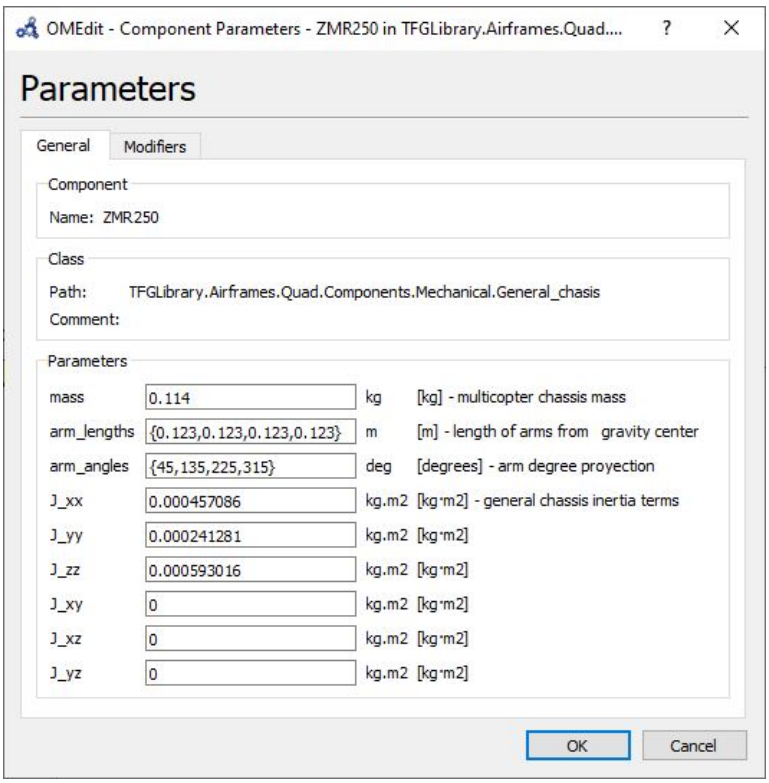


Figura B.4: Ventana de propiedades de chasis en OpenModelica